Adaptive Particle Swarm Optimization*

Zhi-hui Zhan and Jun Zhang

Department of Computer Science, Sun Yat-sen University, China
junzhang@ieee.org

Abstract. This paper proposes an adaptive particle swarm optimization (APSO) with adaptive parameters and elitist learning strategy (ELS) based on the evolutionary state estimation (ESE) approach. The ESE approach develops an 'evolutionary factor' by using the population distribution information and relative particle fitness information in each generation, and estimates the evolutionary state through a fuzzy classification method. According to the identified state and taking into account various effects of the algorithm-controlling parameters, adaptive control strategies are developed for the inertia weight and acceleration coefficients for faster convergence speed. Further, an adaptive 'elitist learning strategy' (ELS) is designed for the best particle to jump out of possible local optima and/or to refine its accuracy, resulting in substantially improved quality of global solutions. The APSO algorithm is tested on 6 unimodal and multimodal functions, and the experimental results demonstrate that the APSO generally outperforms the compared PSOs, in terms of solution accuracy, convergence speed and algorithm reliability.

1 Introduction

Particle swarm optimization (PSO) is one of the swarm intelligence (SI) algorithms that was first introduced by Kennedy and Eberhart in 1995 [1], inspired by swarm behaviors such as birds flocking and fishes schooling. Since its inception in 1995, PSO has been seen rapid development and improvement, with lots of successful applications to real-world problems [2].

Attempts have been made to improve the PSO performance in recent years and a few PSO variants have been proposed. Much work focused on parameters settings of the algorithm [3,4] and on combining various techniques into the PSO [5,6,7]. However, most of these improved PSOs manipulate the control parameters or hybrid operators without considering the varying states of evolution. Hence, these operations lack a systematic treatment of evolutionary state and still sometimes suffer from deficiency in dealing with complex problems.

This paper identifies and utilizes the distribution information of the population to estimate the evolutionary states. Based on the states, the adaptive

* This work was supported by NSF of China Project No.60573066 and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry, P.R. China.

M. Dorigo et al. (Eds.): ANTS 2008, LNCS 5217, pp. 227–234, 2008.

control parameters strategies are developed for faster convergence speed and the elitist learning strategy (ELS) is carried out in a convergence state to avoid the probability of being trapped into local optima. The PSO is thus systematically extended to adaptive PSO (APSO), so as to bring about outstanding performance when solving global optimization problems.

The rest of this paper is organized as follows. In Section 2, framework of PSO will be described. Then Section 3 presents the evolutionary state estimation (ESE) approach in details and develops the ESE enabled adaptive particle swarm optimization (APSO) through an adaptive control of PSO parameters and an adaptive elitist learning strategy (ELS). Section 4 compares this APSO algorithm against some various existing PSO algorithms using a number of test functions. Conclusions are drawn in Section 5.

2 Particle Swarm Optimization

In PSO, a swarm of particles are introduced to represent the solutions. Each particle i is associated with two vectors, the velocity vector $\boldsymbol{V_i} = [v_i^1, v_i^2, ..., v_i^D]$ and the position vector $\boldsymbol{X_i} = [x_i^1, x_i^2, ..., x_i^D]$. During an iteration, the fitness of particle i will first be evaluated at its current position. If the fitness is better than that of $pBest_i$, defined as the best solution that the i^{th} particle has achieved so far, then $pBest_i$ will be replaced by the current solution. Following updating all $pBest_i$, the algorithm selects the best $pBest_i$ among the entire swarm as the global best, denoted as *gBest*. Then, the velocity and position of every particle are updated as (1) and (2)

$$v_i^d = \omega \times v_i^d + c_1 \times rand_1^d \times (pBest_i^d - x_i^d) + c_2 \times rand_2^d \times (gBest^d - x_i^d). \quad (1)$$

$$x_i^d = x_i^d + v_i^d. \quad (2)$$

where ω is inertia weight linearly decreasing from 0.9 to 0.4 [3] and c_1, c_2 are acceleration coefficients that are conventionally set to a fixed value 2.0; $rand_1^d$ and $rand_2^d$ are two independently generated random numbers within the range [0, 1] for the d^{th} dimension. Then the algorithm goes to iteration, until a stop condition is met.

Given its simple concept, PSO has been applied in many fields concerning optimization and many researchers have attempted to improve the performance, with variants of PSOs proposed [2].

On the concerns of parameters study, Shi and Eberhart introduced the linearly decreasing inertia weight [3]. Also, Ratnaweera et al. [4] has proposed a linearly time-varying values method for both acceleration coefficients, namely HPSO-TVAC, with a larger c_1 and a smaller c_2 at the beginning and gradually decreasing c_1 whilst increasing c_2 during the running time.

What is more, different techniques such like the selection [5], mutation [4] introduced from GAs have been merged into original PSO to improve the performance. By the inspiration of biology, some researchers introduced niche technology [6] and speciation technology [7] into PSO on the purpose of avoiding the swarm crowding too close and locating as many optimal solutions as possible.

3 Particle Swarm Optimization

3.1 Evolutionary State Estimation

The evolutionary state estimation (ESE) approach in this paper will use not only the fitness information of individuals, but also the population distribution information of the swarm. The evolutionary state in each generation is determined by a fuzzy classification method controlled by an *evolutionary factor f*. These techniques and the estimation process are detailed in the following steps.

Step 1: At current position, calculate the mean distance of particle i to all the other particles by (3)

$$d_i = \frac{1}{N-1} \sum_{j=1, j \neq i}^{N} \sqrt{\sum_{k=1}^{D} (x_i^k - x_j^k)^2}. \tag{3}$$

where N and D are the population size and dimension, respectively.

Step 2: Compare all d_i's and determine the maximal distance $d_{\max}$ and the minimal distance $d_{\min}$. Denote d_i of the global best particle by d_g. Define an *evolutionary factor* f as (4)

$$f = \frac{d_g - d_{\min}}{d_{\max} - d_{\min}} \in [0, 1]. \tag{4}$$

which is set to 1 if $d_{\max}$ is equal to $d_{\min}$, and is also initialized to 1 when the algorithm starts.

Step 3: Classify the value of f through fuzzy set membership functions, as shown in Fig. 1(a), and hence determine the current evolutionary state into one of the four different states, say, convergence, exploitation, exploration and jumping-out states. These membership functions are designed empirically and are according to the intuitions that f is relative large in the exploration or jumping-out state and is relative small in the exploitation or convergence state. Since the functions are likely to overlap, the expected oscillation sequence of S_i, such as $S_3 => S_2 => S_1 => S_4 => S_3 =>$, may be further used to ascertain the classification.

3.2 Adaptive Strategies for Parameters

The inertia weight ω is used to balance the global and local search abilities, and was suggested to linearly decrease from 0.9 to 0.4 with generation [3]. However, it is not necessarily proper to decrease purely with time. Hence, in this paper, the value of the ω is adaptively adjusted by the mapping $\omega(f) : \Re \to \Re$ as (5).

$$\omega(f) = \frac{1}{1 + 1.5e^{-2.6f}} \in [0.4, 0.9], \forall f \in [0, 1]. \tag{5}$$

Note that, with the mapping function, ω now changes with f, with large value in exploration state and small value in exploitation state, but not purely with

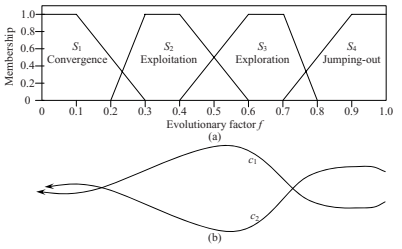

Fig. 1. (a) Fuzzy membership functions for different evolutionary states; (b) Adaptation of the acceleration coefficients according to ESE with the ideal sequence of states $S_3 => S_2 => S_1 => S_4 => S_3 =>$

time or with the generation number. Hence, the adaptive inertia weight is expected to be changed efficiently according to the evolutionary states. Since f is initialized to 1, ω is therefore initialized to 0.9 in this paper.

Acceleration coefficients c_1 and c_2 are also important for the exploration and exploitation abilities. In [4], the values of c_1 and c_2 are dynamic changed with time, with larger c_1 and smaller c_2 at the beginning for better exploration and smaller c_1 with larger c_2 at the end for better convergence. Based on the effect of these two parameters, this paper adaptively adjusts them according to the strategies as in Table 1 for different evolutionary states.

Table 1. Strategies for tuning the values of c_1 and c_2

Strategies	States	c_1	c_2
Strategy 1	Exploration	Increase	Decrease
Strategy 2	Exploitation	Slight increase	Slight decrease
Strategy 3	Convergence	Slight increase	Slight increase
Strategy 4	Jumping-out	Decrease	Increase

These strategies share the common attempts with [4] to control the acceleration coefficients dynamic. However, the strategies in this paper are according to the evolutionary states and are expected to be more reasonable and warrantable. The values of c_1 and c_2 are initialized to 2.0 and gradually change as illustrated in Fig. 1(b). With larger c_1 in the exploration state and larger c_2 in the convergence state, the algorithm will balance the global and local search ability adaptively. What is more, a larger c_2 with a smaller c_1 in the jumping-out state can make the swarm in local optimal region separate and fly to the new better region as fast as possible.

The generational change is as (6) where δ is a uniformly generated random value in the range [0.05, 0.1], as indicated by the empirical study. It should be noticed that we use 0.5δ in the strategies 2 and 3 where "Slight" changes are used.

$$c_i(g+1) = c_i(g) \pm \delta, i = 1, 2. \tag{6}$$

What is more, the values of c_1 and c_2 are clamped in range [1.5, 2.5] and their sum is clamped within [3.0, 4.0]. If the sum exceeds the bound, the values of c_1 and c_2 are adjusted by sliding scale.

3.3 Elitist Learning Strategy for *gBest*

The ESE enabled adaptive parameters are expected to bring faster convergence speed to the PSO algorithm. Nevertheless, when the algorithm is in a convergence state, for the *gBest* particle, it has no other exemplars to follow. So the standard learning mechanism does not help *gBest* escape from the current optimum if it is local. Hence, an elitist learning strategy (ELS) is developed in this paper to give momentum to the *gBest* particle. The ELS randomly chooses one dimension of *gBest*'s historical best position, denoted by p^d, and assigns it with momentum to move around. For this, a learning strategy through Gaussian perturbation as (7)

$$p^d = p^d + (X^d_{\max} - X^d_{\min}) \times Gaussian(\mu, \sigma^2). \tag{7}$$

within the saturation limits $[X^d_{\min}, X^d_{\max}]$ can be applied. Here, Gaussian(μ,σ^2) represents Gaussian distribution with a mean μ=0 and a time-varying standard deviation as (8)

$$\sigma = \sigma_{\max} - (\sigma_{\max} - \sigma_{\min}) \times (g/G). \tag{8}$$

where $\sigma_{\max}$=1.0 and $\sigma_{\min}$=0.1 as indicated by the empirical study. It is should be noted that, the new position will be accepted if and only if its fitness is better than the current *gBest*.

4 Experimental Tests and Comparisons

4.1 Testing Functions and Tested PSOs

Six benchmark functions listed in Table 2 are used for the experimental tests. These test functions are widely adopted in benchmarking optimization algorithms [8]. The first 3 are unimodal functions, and the second 3 are complex multimodal functions with a large number of local minima. For details of these functions, refer to [8].

Table 2. Six test functions used in comparison

Test function	n	Search Space	$f_{\min}$	Acceptance
$f_1 = \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0	0.01
$f_2 = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	$[-10, 10]^n$	0	0.01
$f_3 = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	30	$[-10, 10]^n$	0	100
$f_4 = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	30	$[-500, 500]^n$	-12569.5	-10000
$f_5 = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	30	$[-5.12, 5.12]^n$	0	50
$f_6 = -20\exp(-0.2\sqrt{1/n \sum_{i=1}^{n} x_i^2})$ $- \exp(1/n \sum_{i=1}^{n} \cos 2\pi x_i) + 20 + e$	30	$[-32, 32]^n$	0	0.01

The PSO-IW [3] and HPSO-TVAC [4] algorithms are used here for comparison because PSO-IW aims to improve the parameter inertia weight while HPSO-TVAC is improved PSO, by improving the acceleration coefficients. The parameters of these PSOs are set according to the literatures [3,4], and the parameters of APSO are as descriptions above. For a fair comparison among all the three PSOs, they are tested using the same population size of 20, and the same maximal number of 2.0×10^5 function evaluations (FEs) for each test function. Each function is simulated 30 trials independently and their mean values are used in the comparisons.

4.2 Results Comparisons and Discussions

The performance of every PSO is compared in Table 3, in terms of the mean and standard deviation of the solutions obtained by each algorithm.

Table 3. Results of variant PSOs on six test functions

f	PSO-IW	HPSO-TVAC	APSO
f_1	$1.98 \times 10^{-53} \pm 7.08 \times 10^{-53}$	$3.38 \times 10^{-41} \pm 8.50 \times 10^{-41}$	$1.45 \times 10^{-150} \pm 5.73 \times 10^{-150}$
f_2	$2.51 \times 10^{-34} \pm 5.84 \times 10^{-34}$	$6.90 \times 10^{-23} \pm 6.89 \times 10^{-23}$	$5.15 \times 10^{-83} \pm 1.44 \times 10^{-83}$
f_3	28.1 ± 24.6	13.0 ± 16.5	2.84 ± 3.27
f_4	-10090.16 ± 495	-10868.57 ± 289	$-12569.5 \pm 5.22 \times 10^{-11}$
f_5	30.7 ± 8.68	2.39 ± 3.71	$5.80 \times 10^{-15} \pm 1.01 \times 10^{-14}$
f_6	$1.15 \times 10^{-14} \pm 2.27 \times 10^{-15}$	$2.06 \times 10^{-10} \pm 9.45 \times 10^{-10}$	$1.11 \times 10^{-14} \pm 3.55 \times 10^{-15}$

The comparisons in Table 3 show that, when solving unimodal problems, APSO offers the best performance on all the test functions. The fact that the APSO can obtain better solutions on unimodal functions indicates that its adaptive nature indeed offers a faster convergence speed. What is more, APSO outperforms other PSOs on the optimization of all the complex multimodal functions f_4-f_6 as the results presented in Table 3. The advantages are more evident while solving the much more complex problems as Schwefel's function (f_4) and the Rastrigin's function (f_5). This suggests that the APSO has the ability of jumping out local optimal and achieve the global optimum efficiently.

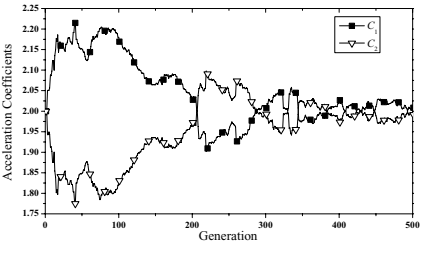

Fig. 2. Adaptive acceleration coefficients during the running time on f_5

Table 4. Mean FEs to reach acceptable solutions and successful ratio

f	PSO-IW	HPSO-TVAC	APSO
f_1	105695(100.0%)	30011(100.0%)	7074(100.0%)
f_2	103077(100.0%)	31371(100.0%)	7900(100.0%)
f_3	101579(100.0%)	33689(100.0%)	5334(100.0%)
f_4	90633(100.0%)	44697(100.0%)	5159(100.0%)
f_5	94379(56.7%)	7829(100.0%)	3531(100.0%)
f_6	110844(96.7%)	52516(100.0%)	40736(100.0%)
Mean Reliability	92.2%	100.0%	100.0%

In order to track the change of acceleration coefficients, Fig. 2 plots the curves of c_1 and c_2 on function f_5 for the first 500 generations. Fig. 2 shows that c_1 is increasing whilst c_2 is decreasing for a number of generations at the beginning because the population is exploring for the optimum. Then c_1 and c_2 reverse their change directions when exploiting for convergence. The jumping out state can also be detected where the value of c_2 increases, c_1 decreases. The search behavior indicates that APSO algorithm has indeed identified the evolutionary states and can adaptively adjust the parameters for better performance.

Table 4 reveals that the APSO offers a generally faster convergence speed, using a small number of function evaluations (FEs) to reach an acceptable solution. For example, tests on f_1 show that the average numbers of FEs of 105695 and 30011 are needed by the PSO-IW and HPSO-TVAC, respectively, to reach an acceptable solution. However, the APSO uses only 7074 FEs to reach the solution. Table 4 also reveals that the APSO offers a generally highest percentage of trials reaching acceptable solutions and the highest reliability averaged over all the test functions.

While the APSO uses identified evolutionary states to adaptively control the algorithm parameters for a faster convergence, it also performs elitist learning in the convergence state to avoid possible local optima. In order to quantify the significance of these two operations, the performance of the APSO without parameters adaptation or elitist learning was tested. Results of mean values on 30 independent trials are presented in Table 5.

Experimental results in Table 5 show that with elitist learning only and without adaptive control of parameters, the APSO can still deliver good solutions to multimodal functions (although with a much lower speed, such a lower

Table 5. Merits of parameter adaptation and elitist learning

f	APSO with Both Adaptation & Learning	APSO Without Parameters Adaptation	APSO Without Elitist Learning	PSO-IW (Standard PSO Without Either)
f_1	1.45×10^{-150}	3.60×10^{-50}	7.67×10^{-160}	1.98×10^{-53}
f_2	5.15×10^{-84}	2.41×10^{-32}	6.58×10^{-88}	2.51×10^{-34}
f_3	2.84	12.75	13.89	28.10
f_4	-12569.5	-12569.5	-7367.77	-10090.16
f_5	5.80×10^{-15}	1.78×10^{-16}	52.73	30.68
f_6	1.11×10^{-14}	1.12×10^{-14}	1.09	1.15×10^{-14}

convergence speed can be reflected by the lower accuracy in solutions to unimodal functions at the end of the search run). On the other hand, the APSO with parameters adaptation only and without an ELS can hardly jump out of local optima and hence results in poor performance on multimodal functions, but it can still solve unimodal problems well. However, both reduced APSO algorithms generally outperform a standard PSO with neither parameters adaptation nor elitist learning, but the full APSO is the most powerful and robust for any given problem. This is most evident in the test result on f_3. These results confirm the hypothesis that adaptive control of parameters speeds up the convergence while elitist learning helps to jump out of local optima.

5 Conclusions

An adaptive particle swarm optimization (APSO) enabled by evolutionary state estimation has been developed in this paper. Experimental results show that the proposed algorithm yields outstanding performance on not only unimodal, but also multimodal function, with faster convergence speed, higher accuracy solutions, and better algorithm reliability. Future work will focus on testing the APSO on a comprehensive set of benchmarking functions and the applications to real-world optimization problems.

References

1. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, pp. 1942–1948 (1995)
2. Li, X.D., Engelbrecht, A.P.: Particle Swarm Optimization: an Introduction and Its Recent Developments. In: Proceedings of the 2007 Genetic Evolutionary Computation Conference, pp. 3391–3414 (2007)
3. Shi, Y., Eberhart, R.C.: A Modified Particle Swarm Optimizer. In: Proceedings of the IEEE World Congress on Computation Intelligence, pp. 69–73 (1998)
4. Ratnaweera, A., Halgamuge, S., Watson, H.: Self-organizing Hierarchical Particle Swarm Optimizer with Time-varying Acceleration Coefficients. J. IEEE Trans. Evol. Comput. 8, 240–255 (2004)
5. Angeline, P.J.: Using Selection to Improve Particle Swarm Optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, Anchorage, AK, pp. 84–89 (1998)
6. Brits, R., Engelbrecht, A.P., van den Bergh, F.: A Niching Particle Swarm Optimizer. In: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolutionary Learning, pp. 692–696 (2002)
7. Parrott, D., Li, X.D.: Locating and Tracking Multiple Dynamic Optima by a Particle Swarm Model Using Speciation. J. IEEE Trans. Evol. Comput. 10, 440–458 (2006)
8. Yao, X., Liu, Y., Lin, G.M.: Evolutionary Programming Made Faster. J. IEEE Trans. Evol. Comput. 3, 82–102 (1999)